

Lecture 14:
Project Proposals &
Finish Transactions

Wednesday, March 11, 2015

Agenda

- Issues with HW #3
- Project proposal feedback
- Continue transactions and start recoveries
- Take Quiz #4

Project Proposals

- **Proposals reviewed**
 - Documents returned with comments
- **Upcoming schedule:**
 - ERD and SQL due this Friday by 11:59pm
 - Submit them to me by email
 - Indicate your group number in subject line
 - Copy all group members on the email
 - Will return them with comments next week
 - Class presentations on 03/30, 04/01, and 04/06
 - Groups 1 – 9 on 03/30
 - Groups 10 – 18 on 04/01
 - Groups 19 – 27 on 04/06
 - Final submissions due on 04/06

Topics Breakdown

Topic	Count
YELP!YELP!YELP!YELP!	11
Other datasets (e.g. *data.gov, freebase.com)	10
TV show & video game	2
Sports	3
Fashion	1

Project Groups

Grp	Members
1	Matthew Egbom, Jewel Langevine, and Lerone Williams
2	Nathan Waters and Nur Ridzuan
3	Steve Franklin, Sadie Sublousky, and Tien-Yu Huang
4	Mills Hill
5	Alexander Crompton and Jacob Rachiele
6	Mitali Sathaye
7	Nikolaj Plagborg-Moller and Fabiana Latorre
8	Hannah Jane DeCiutiis, Kathryn McDermott, and Esther Schenau
9	Khang Pham and Don Pham
10	Alexia Mercado and Cyndia Munoz
11	Thomas Johnson and John Loftin
12	Ross Yudkin, Kurt Probe, and Andrew Chang-Gu
13	Tianxiang Zhang, Xiaolin Lu, and Happy Situ
14	Kaitlin Vanderlaan, Julia Haschke, and Sarah Luna
15	Brian Huang, Sergio Mier, and Jun-Bo Shim
16	Jose Cortez, David Hernandez, and Tara Woolheater
17	Kerri Grier and Chris Oballe
18	Matthew Jones, Thomas Reay, and Brooke Noble
19	Seata Moji and Alexander Thola
20	Hyun Seo and Parth Patel
21	Yifang Peng and Jiannan Zhang
22	Dustin Dies, Sreejon Sen, and John Huynh
23	Cameron Miller, Jorge Paramo, and Kyle Kerr
24	Humza Rashid, Mark Slater, and Matthew Mcnair
25	Robert Mcneil and Zachary Williams
26	Bailey Lund, Kristine Chen, and Irene Jea
27	Damilola Shonaike and Bryan Landes

Project Presentation

- **10 minutes** per project: 7 minutes presentation plus 3 minutes for questions.
- Suggested content:
 - describe the problem
 - describe your approach
 - give short demo
 - discuss unexpected issues or problems
 - discuss possible extensions

Final Project Submission

- A one page report on how the project was implemented and how it works internally
- A brief description of the code that has been written
- A brief description of the experiments you ran to verify the solution
- End-user documentation (instructions and examples on how somebody can use this project)
- Submit all code including dataset and test cases

Success Criteria

- Do a good job of modeling the domain
- Your database should enable the entire analysis pipeline
- Use SQL to compute the visualizations
- Develop a simple app to interact with your database
- If not working with a dataset, populate your database with sample data (20-30 records is sufficient)
- If working with a dataset, choose a strategy to explore the data. Keep track of your work even those queries that don't produce any interesting trends!

Review Questions

- What is a *transaction*?
- What is a *schedule* in concurrency control?
- What is a *serializable schedule*?
- What is a *recoverable* schedule?

Review Questions

- **What is a transaction?**

Ans: A sequence of reads and writes

- **What is a *schedule* in concurrency control?**

Ans: The interleaving of reads and writes by multiple txs that are running concurrently in a database

- **What is a *serializable schedule*?**

Ans: A schedule is *serializable* if its effects are equivalent to the effects of some serial schedule

- **What is a *recoverable* schedule?**

Ans: If every tx commits only after all txs whose changes they read have also committed

Practice: Is this schedule *serializable*?

T1	T2
READ(A)	
	READ(A)
WRITE(B)	
	WRITE(B)
COMMIT	
	COMMIT

Practice: Is this schedule *serializable*?

T1	T2
READ(A)	
	READ(A)
WRITE(B)	
	WRITE(B)
COMMIT	
	COMMIT

Ans: Yes, because its effects are equivalent to those of the serial schedule T1;T2. In both schedules, T1 and T2 read the initial value of A, neither tx reads a value written by the other, and T2 performs the final write of B.

Practice: Is this schedule *recoverable*?

T1	T2
READ(A)	
	READ(A)
WRITE(B)	
	WRITE(B)
COMMIT	
	COMMIT

Practice: Is this schedule *recoverable*?

T1	T2
READ(A)	
	READ(A)
WRITE(B)	
	WRITE(B)
COMMIT	
	COMMIT

Ans: Yes, because there are no dirty reads. In other words, neither tx reads a value that has been written by the other before the commit. Therefore, there's no need to worry about the order in which the txs commit.

Practice: Is this schedule *serializable*?

T1	T2
READ(A)	
	WRITE(A)
READ(A)	
WRITE(B)	
COMMIT	
	COMMIT

Practice: Is this schedule *serializable*?

T1	T2
READ(A)	
	WRITE(A)
READ(A)	
WRITE(B)	
COMMIT	
	COMMIT

Ans: No, because it's not equivalent to either of the two possible serial schedules T1;T2 or T2;T1. It's not equivalent to T1;T2 because T1's second read of A reads the value written by T2, but in the serial schedule T1;T2, T2 would read the initial value of A. It's not equivalent to T2;T1 because T1's first read of A reads the initial value of A, but in the schedule T2;T1 it would read the value written by T2.

Practice: Is this schedule *recoverable*?

T1	T2
READ(A)	
	WRITE(A)
READ(A)	
WRITE(B)	
COMMIT	
	COMMIT

Practice: Is this schedule *recoverable*?

T1	T2
READ(A)	
	WRITE(A)
READ(A)	
WRITE(B)	
COMMIT	
	COMMIT

Ans: No, because T1 performs a dirty read (i.e. it reads a value of A written by T2 before T2 commits) and T1 commits before T2 commits. If T2 were to rollback after T1 commits, the database would be in an inconsistent state. This would happen because the rollback of T2 would undo T2's write of A and yet T1 could have performed an action based on that write (e.g. the write of B could have been based on a dirty read of A).

Causes of Recovery

- Data corruption
- System failures
- Data center outage

Principles in Recovery

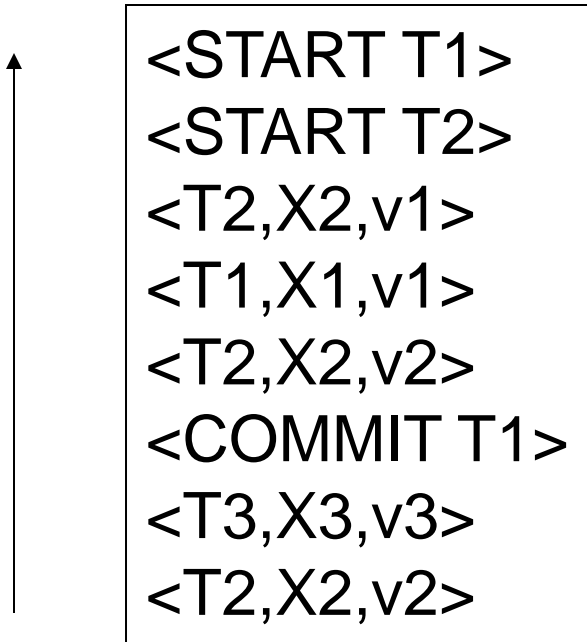
- Write-ahead log = A file that records every action of all running txs
- *Force* log entries to disk
- After a crash, recovery manager reads log entries and finds out exactly which txs were in flight
- Oracle uses two types log files: redo and undo.
- Oracle replays the undo log to undo values of uncommitted txs
- Oracle replays the redo log to redo values of committed txs

Question: why do we need both undo and redo?

Undo Log Entry

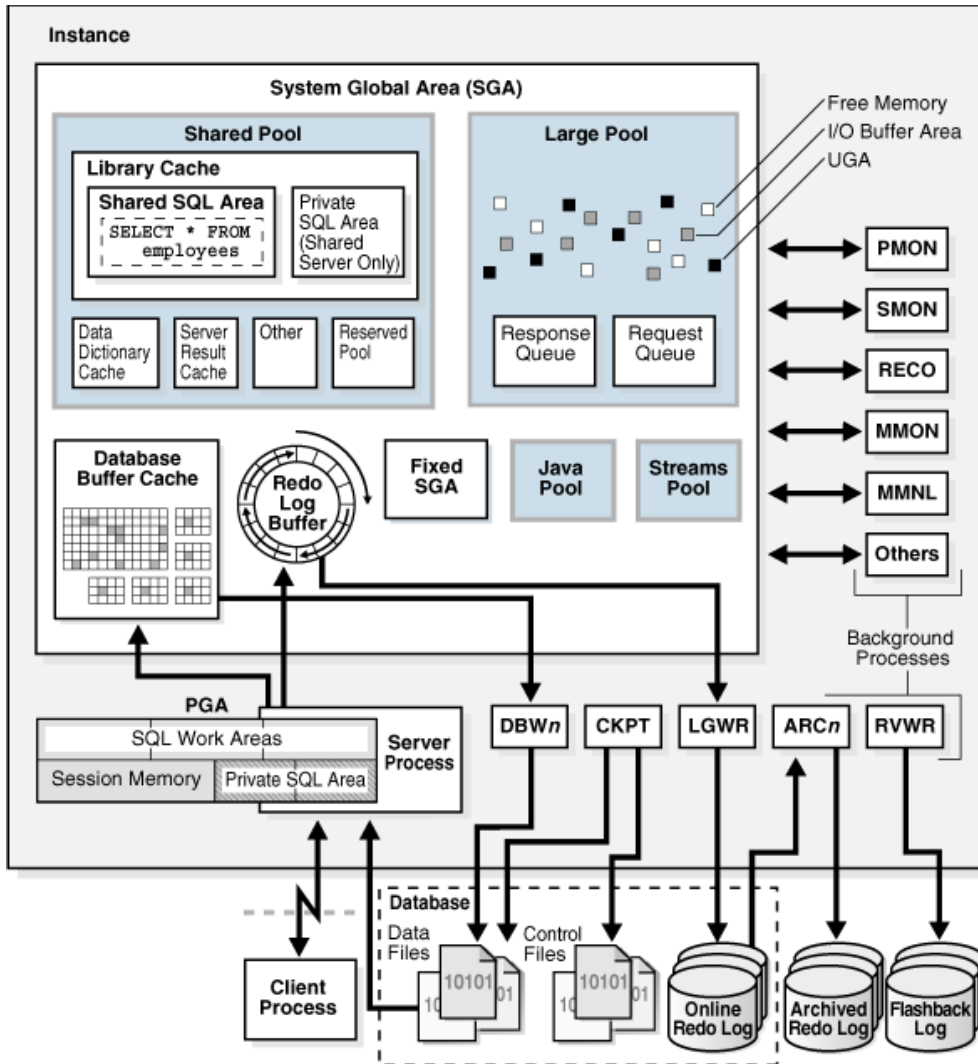
- <START T>
 - transaction T has begun
- <COMMIT T>
 - T has committed
- <ROLLBACK T>
 - T has rolled-back
- <T,X,v>
 - T has updated record X, and its old value was v

Recovery with Undo Log



Which updates need to be undone?

Why Undo is not enough



Source: [Oracle Database Concepts Guide](#) (version 12c)


View of Memory versus Disk

Tx operation	Mem	Disk A	Disk B
READ(A,x)	x=8	8	8
x:=x*2	x=16	8	8
WRITE(A,x)	x=16	8	8
READ(B,y)	y=8	8	8
y:=y*2	y=16	8	8
WRITE(B,y)	y=16	8	8
COMMIT(A)	x=16	16	8
COMMIT(B)	y=16	16	16

Redo Log Entry

- $\langle \text{START } T \rangle$ = transaction T has begun
- $\langle \text{COMMIT } T \rangle$ = T has committed
- $\langle \text{ROLLBACK } T \rangle$ = T has rolled-back
- $\langle T, X, v \rangle$ = T has updated record X, and its new value is v

Recovery with Redo Log



<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>

Which updates need to be redone?

Quiz #4

T1	T2	T3
READ(A)	READ(B)	READ(B)
READ(B)	READ(C)	READ(C)
WRITE(A)	WRITE(B)	WRITE(C)
COMMIT	ROLLBACK	COMMIT

Q1: Give a *serial* schedule for T1, T2 and T3.

Q2: Give a *serializable* schedule for T1, T2 and T3.

Q3: Give a *non-recoverable* schedule for T1, T2 and T3.

Q4: Give a *recoverable* schedule for T1, T2 and T3.

Next class

- Project feedback on ERD and SQL
- Indexes